Dossier d’analyse

# Extraire-Transformer-Charger

*Fichiers : models/read\_files.py et EDA\_df.ipynb*

On commence l’analyse par une exploration des données. À cette fin, les fichiers sont lus en utilisant la fonction read\_csv() de la bibliothèque pandas.

Il y a deux fichiers provenant de deux plateformes différentes :

* un fichier contenant les logs, sur lesquels nous souhaitons baser notre modèle et nos prédictions,
* un fichier de notes, avec les résultats des apprenants que nous voulons prédire.

## Les données *logs*

Un fichier contentant les logs, qui représentent les traces d’activités de chaque apprenant d’un cours sur la plateforme ARCHE.

Les champs sont les suivants :

|  |  |  |  |
| --- | --- | --- | --- |
| **Champ** | **Description** | **Format** | **Spécifications** |
| heure | Horodatage de l’activité | Datetime  Année-mois-jour heure:minutes:secondes | Dates entre 23/02/2023 et 14/12/2024 |
| pseudo | Id de l’apprenant | Int représentant l’apprenant après anonymisation | Chiffre entre 5 et 1000 |
| contexte | Ressource objet de l’activité | Object (String: string) | 46 valeurs uniques |
| composant | Type d’activité | Object (string) | 17 valeurs uniques |
| evenement | Précisions sur l’activité | Object (string) | 61 valeurs uniques |

Une première analyse montre que :

* Le fichier contient 29 006 lignes et 5 colonnes
* Il n’y pas de valeurs manquantes.

On constate qu’il y a des pseudos dans les logs qui n’ont pas de correspondance dans les notes. Les lignes avec ces pseudos sont enlevées.

La colonne « contexte » parait être un ensemble de deux éléments, séparés par un « : ». La colonne est découpée en deux colonnes : « contexte\_general » et « specification ». La colonne « contexte » est devenue obsolète et est alors supprimée.

Pour pouvoir travailler avec les notions jours et heures, on sépare également la colonne « heure » en deux partie, une colonne « jour » avec les dates et une colonne « heures » avec l’heure exacte. La première colonne est convertie en datetime, la deuxième en time. On garde la colonne « heure » pour faire des opérations avec pandas, qui ne reconnait pas le format « time ».

On obtient un dataframe qui contient 27 771 lignes et 8 colonnes.

|  |  |  |  |
| --- | --- | --- | --- |
| **Champ** | **Description** | **Format** | **Spécifications** |
| heure | Horodatage de l’activité | Datetime  Année-mois-jour heure:minutes:secondes | Dates entre 23/02/2023 et 14/12/2024 |
| pseudo | Id de l’apprenant | Int représentant l’apprenant après anonymisation | Chiffre entre 5 et 1000 |
| composant | Type d’activité | Object (string) | 13 valeurs uniques |
| evenement | Précisions sur l’activité | Object (string) | 27 valeurs uniques |
| contexte\_general | Ressource objet de l’activité | Object (string) | 8 valeurs uniques |
| specification | Spécification sur la ressource | Object (string) | 45 valeurs uniques |
| jour | Date | Datetime  Année-mois-jour | Dates entre 23/02/2023 et 14/12/2024 |
| heures | Heure | Time (pas reconnu en pandas)  Heure:minutes:secondes | Dans pandas considéré comme object |

## Les données *notes*

Un fichier contentant les notes obtenues par les apprenants.

|  |  |  |  |
| --- | --- | --- | --- |
| **Champ** | **Description** | **Format** | **Description** |
| pseudo | Id de l’apprenant | Chiffre représentant l’apprenant après anonymisation | Forme le lien avec les données logs |
| note | Note obtenue par l’apprenant | String à cause de tirets (-) présents | Entre « - » et 19 |

Une première analyse montre que :

* Le fichier contient 80 lignes et 2 colonnes
* Il n’y pas de valeurs manquantes

Les notes sont interprétées comme des objets (string), à cause des tirets qui sont présents. Afin de les convertir en entiers, on remplace les tirets par des zéros.

Un histogramme de la distribution des notes montre que le taux de réussite est assez faible, avec seulement 30% des apprenants obtenant une note de 10 au plus. Les notes les plus fréquentes sont 2 et 3, suivis par 4 et 8. On suppose que les apprenants sont notés sur 20.

Pour le suivi des analyses les pseudos qui n’ont pas de correspondance dans les logs sont supprimés.

On obtient alors un dataframe avec 78 lignes et 2 colonnes.

|  |  |  |  |
| --- | --- | --- | --- |
| **Champ** | **Description** | **Format** | **Description** |
| pseudo | Id de l’apprenant | Entier représentant l’apprenant après anonymisation | Forme le lien avec les données logs |
| note | Note obtenue par l’apprenant | Entier | Entre 0 et 19 |

On constate que le dataframe logs n’est pas en bon format pour le développement d’un modèle de machine learning. La construction d’un nouveau dataframe avec des indicateurs utilisables est alors nécessaire.

# Feature engineering

## Calcul des indicateurs

*Fichier : controllers/features\_creation.py et EDA\_logs.ipynb*

Les indicateurs (« features ») suivants sont calculés :

1. Le nombre d’actions réalisées par chaque apprenant
2. La moyenne du nombre d’actions réalisées par jour par chaque apprenant
3. Le nombre d’actions maximal réalisées par jour par chaque apprenant
4. La variabilité dans le nombre d’actions réalisées par jour par chaque apprenant
5. Le nombre de jours avec activité
6. La différence (en jours) entre la première et la dernière activité
7. La constance d’activité
8. Le pourcentage d’actions réalisées pendant le week-end
9. La période moyenne d’activité par jour
10. Le pourcentage d’actions réalisées pendant la nuit
11. Le pourcentage d’actions réalisées le matin
12. Le pourcentage d’actions réalisées l’après-midi
13. Le pourcentage d’actions réalisées le soir
14. Le nombre de composants utilisés par chaque apprenant
15. Le nombre d’événements utilisés par chaque apprenant
16. Le nombre de contextes générales utilisés par chaque apprenant
17. Le nombre d’actions pour chaque composant par apprenant
18. Le nombre d’actions pour chaque événement par apprenant
19. Le nombre d’actions pour chaque contexte général par apprenant
20. Le composant le plus utilisé par apprenant
21. L’événement le plus utilisé par apprenant
22. Le contexte général le plus utilisé par apprenant

On obtient alors un dataframe avec 78 lignes et 66 colonnes. Chaque ligne dans ce nouveau dataframe correspond à une ligne dans le dataframe des notes. Il n’y a pas de valeurs manquantes.

Les colonnes : *'nb\_actions', 'moyenne\_nb\_actions', 'nb\_jours\_avec\_action', 'std\_actions\_par\_jour', 'tempsdiff\_jours', 'constance\_activite', 'activite\_moyenne\_par\_jour\_min', 'pourcentage\_activite\_nuit', 'pourcentage\_activite\_matin', 'pourcentage\_activite\_aprem', 'pourcentage\_activite\_soir', 'pct\_weekend', 'nb\_contexte', 'nb\_specification', 'nb\_composant', 'contexte\_Bloc\_notes\_collaboratif', 'contexte\_Cours', 'contexte\_Devoir', 'contexte\_Fichier', 'contexte\_Forum', 'contexte\_Présence', 'contexte\_Questionnaire', 'contexte\_Wooclap', 'composant\_Commentaires', 'composant\_Devoir', 'composant\_Fichier', 'composant\_Questionnaire', 'composant\_Rapport\_du\_participant', 'composant\_Rapport\_d\_ensemble', 'composant\_Remises\_de\_fichiers', 'composant\_Système', 'composant\_Visites\_guidées', 'nb\_evenement', 'evenement\_Achèvement\_d\_activité\_modifié', 'evenement\_Confirmation\_de\_suppression\_de\_travail\_consultée', 'evenement\_Cours\_consulté', 'evenement\_Discussion\_consultée', 'evenement\_Formulaire\_de\_remise\_consulté', 'evenement\_Liste\_d\_utilisateurs\_consultée', 'evenement\_Module\_de\_cours\_consulté', 'evenement\_Profil\_utilisateur\_consulté', 'evenement\_Questionnaire\_poursuivi', 'evenement\_Questionnaire\_soumis', 'evenement\_Rapport\_de\_cours\_utilisateur\_consulté', 'evenement\_Rapport\_de\_session\_consulté', 'evenement\_Rapport\_des\_réponses\_individuelles\_consulté', 'evenement\_Statut\_de\_présence\_renseigné\_par\_l\_étudiant', 'evenement\_Statut\_du\_travail\_remis\_consulté', 'evenement\_Travail\_de\_devoir\_créé', 'evenement\_Travail\_de\_devoir\_modifié', 'evenement\_Utilisateur\_évalué', 'top\_contexte\_Cours', 'top\_contexte\_Devoir', 'top\_contexte\_Fichier', 'top\_contexte\_Présence', 'top\_composant\_Fichier', 'top\_composant\_Présence', 'top\_composant\_Système', 'top\_evenement\_Achèvement\_d\_activité\_modifié', 'top\_evenement\_Cours\_consulté', 'top\_evenement\_Module\_de\_cours\_consulté', 'top\_evenement\_Rapport\_de\_session\_consulté'*

**Distribution des features**

La plupart (54) des colonnes ont des entiers comme valeurs. Il y a 9 colonnes avec des floats et trois colonnes sont catégorielles. Les distributions des features sont diverses, mais ils ont tous au moins deux valeurs différentes.

**Relation features et target**

Il n’y a pas de relation significante entre les notes et les variables catégorielles.

Les corrélations entre les notes et les variables int et float varient entre -0,18 et 0,26. Il y a une corrélation positive (Pearson) significante entre les notes et :

* Evenement\_Fichier déposé
* Evenement\_Travail de devoir remis
* Composant\_Remises de fichiers
* Evenement\_Formulaire de remise consulté
* Evenement\_Travail de devoir modifié
* Evenement\_Travail de devoir créé

Les autres corrélations sont non-significatives.

**Relations feature x feature**

Il y a des très fortes corrélations entre les 6 features ayant une corrélation significative avec le target. Il y a également des corrélations fortes entre plusieurs autres features, ce qui n’est pas étonnante vue que les colonnes « contexte », « evenement » et « composant » sont intrinsèquement fortement liées et forme la base de plusieurs features. Nos analyses futures vont alors devoir prendre en compte cette multicollinearité.

**Suppression des corrélations égales à 1**

Cependant, on constate plusieurs variables qui sont parfaitement corrélée l’un à l’autre (coefficient de corrélation égal à 1). Vue que ces variables contiennent exactement la même information, on va supprimer une colonne de chaque pair. Ainsi 14 variables sont supprimées.

## Feature transformation

*Fichier : controllers/preprocessing.py*

**Train-test split**

Nous séparons le jeu de données en deux : une partie que nous allons utiliser pour l’entrainement du modèle (le trainset) et une partie que nous allons utiliser pour évaluer la performance de notre modèle obtenu (le testset). Nous choisissons une répartition de 80/20%, qui est le plus commun.

**Encoder**

Les 3 variables de type object sont catégorielles. Afin de pouvoir utiliser ces variables dans le développement d’un modèle de machine learning, on va devoir les transformer en valeurs numériques, c’est-à-dire, les encoder. On a des variables catégorielle (sans ordre significatif) avec plusieurs catégories, donc on utilise la méthode OneHot pour encoder les catégories dans plusieurs colonnes appelé dummies.

Le dataframe contient maintenant que des valeurs numériques.

**Normalisation des variables**

Afin de garder les variables binaires telles qu’elles sont, une normalisation MinMax est utilisée.

*Formula : X\_std = (X - Xmin) / (Xmax - Xmin)*

De cette manière toutes les features prennent une valeur entre 0 et 1 et les variables binaires gardent leurs valeurs initiales.

## Feature selection

*Fichier : controllers/feature\_selection.py*

A cause de la forte multicollinearité entre plusieurs features, et le grand nombre de features comparé au nombre d’individus, nous ne pouvons pas inclure toutes les features.

Nous avons développé deux fonctions différentes qui permettent de sélectionner les features les plus pertinentes selon deux manières différentes. La première est appelée *forward\_feature\_selection*. Cette fonction va tester plusieurs modèles de régression linéaire, testant tous les features une par une et à chaque boucle choisissant la feature avec le meilleur R² adjusted pour l’ajouter à la formule. La boucle principale s’arrête dès qu’il n’y a plus d’amélioration en ajoutant plus de features. Le taux permet de spécifier qu’un modèle doit améliorer le R² adjusted avec une certaine quantité, sinon un modèle plus simple avec moins de features est préféré. Par default un taux de 0.001 est utilisé, mais peut facilement être changé.

Une deuxième fonction utilise la méthode Lasso qui va également entrainer un modèle de régression linéaire (en réalité plusieurs, parce que nous utilisons LassoCV avec 5 subsets pour trouver la meilleur valeur alpha). La régularisation L1 utilisé par la méthode Lasso permet de réduire certains coefficients à zéro. Cette opération réduit le surapprentissage en choisissant les features les plus importantes. Le paramètre alpha détermine l’amont de régularisation (alpha plus petit = faible régularisation = peu de coefficients mis à zéro ; alpha plus grand = forte régularisation = plus de coefficients mis à zéro).

Formule : min ∣∣y−Xβ∣∣22​ + α∣∣β∣∣1​

# Modèles

*Fichier : models/model.py*

## Régression linéaire

Le premier modèle testé est une régression linéaire multiple. Nous allons tester deux modèles de régression linéaire différents, celui présent dans le package de statistiques statsmodels (module formula.api) comme ols (Ordinary Least Squares) et celui utilisant le package scikit-learn (module linear\_model). Dans les deux cas, le modèle va essayer de minimiser la somme des différences au carré entre les valeurs observées de la cible et les valeurs prédites. Les deux versions donnant des résultats identiques, le package statsmodels est gardé pour la suite de l’analyse.

La régression linéaire est une méthode statistique utilisée pour modéliser la relation entre une variable dépendante (ou target) et une ou plusieurs variables indépendantes (ou features). L'objectif est de trouver le meilleur hyperplan qui minimise la somme des carrés des différences entre les valeurs observées et les valeurs prédites par le modèle. En termes simples, la régression linéaire cherche à ajuster un hyperplan aux données de manière à ce que la distance entre les points de données réels et les points de données prévus soit la plus petite possible.

Une comparaison des résultats des fonctions de deux librairies montre que les deux sont équivalentes. Pour la suite on ne garde qu’un des deux manières. Pour faciliter la programmation, nous choisissons celui de scikit-learn, une librairie qui contient des fonctions pour entrainer d’autres modèles de machine learning.

## Autres modèles

Les deux autres modèles testés sont des modèles d’ensemble. Le modèle Random Forest calcule des arbres de décision pour prédire la valeur du target. Les données sont divisées en subsets et pour chaque subset un arbre de décision est entrainé indépendamment. Utilisant le principe du bagging, la prédiciton final est obtenu en calculant la moyenne de toutes les prédictions individuelles. C’est un modèle robuste qui est peu sensible au surapprentissage.

Le modèle Adaboost, aussi un modèle d’ensemble, est intéressant parce qu’il utilise le principe de boosting (apprentissage séquentiel). A chaque itération, un model de base est entrainé. Ce modèle de base est en général assez faible. Dans les prochaines itérations, les nouveaux modèles vont attribuer plus de poids aux erreurs du modèle antérieur. La prédiction finale est une somme pondérée de tous les modèles calculés. Grace au « *wisdom of the crowd* », cette combinaison de plusieurs prédictions faibles, donne souvent une bonne performance. En scikit-learn un arbre de décision avec 3 niveaux est proposé par défaut comme estimateur de base.

# Evaluation

*Fichier : models/model.py*

## Métriques utilisées

Une première métrique utilisée pour évaluer la capacité prédictive du modèle sur le jeu test est le RMSE (*root mean square error*) défini comme la racine carrée de la moyenne des écarts entre prédiction et donnée test au carré. Cette métrique permet d’estimer l’écart moyen entre la note prédite et la note actuelle d’un étudiant. Le fait de prendre la racine permet d’interpréter les valeurs qui vont alors reprendre l’unité initiale du target. Une deuxième métrique utilisée est le coefficient de détermination (R²). Cette métrique rapporte la proportion de variation des données réponses expliquées par le modèle.

En Python :

*RMSE = np.sqrt(np.mean((pred - y)\*\*2))*

*R² = 1 - (((y - pred)\*\* 2).sum() / ((y - y.mean()) \*\* 2).sum())*

En plus de la dérivation de métriques, la qualité des modèles est évaluée visuellement à partir de 3 graphiques :

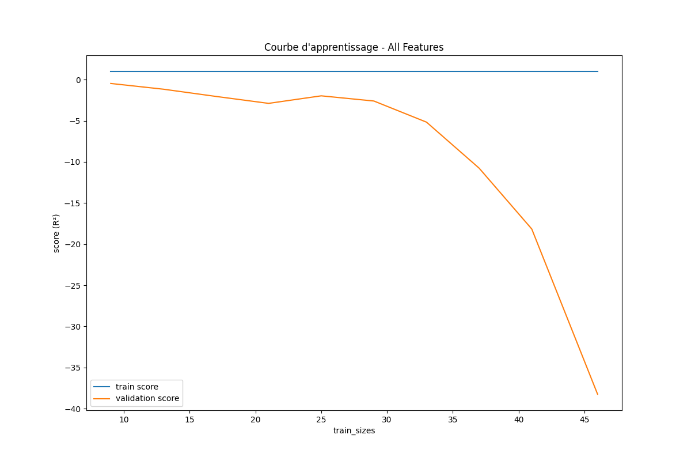
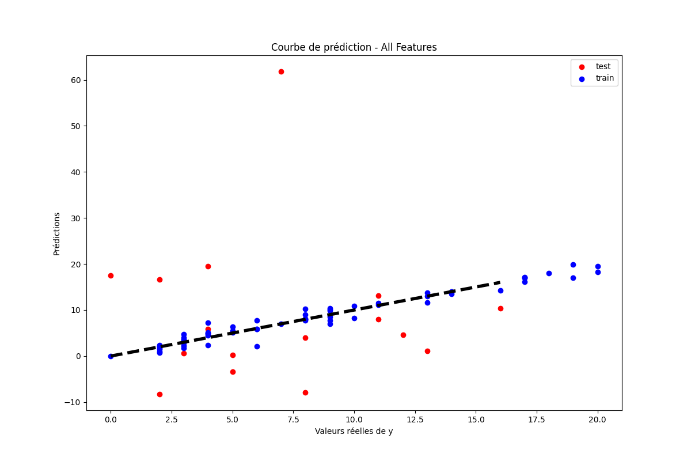
1. Les résidus du modèle en fonction des prédictions du modèle afin d’évaluer si les erreurs sont uniformément distribuées le long du gradient des notes ou si des patrons apparaissent (p.ex. plus forte variabilité pour des notes élevées)
2. Le nuage de points des prédictions contre les notes observées, un modèle performant mène à des points alignés sur la droite 0 :1
3. Le courbe d’apprentissage permettant de visualiser la performance du modèle en fonction de quantité de données, qui permet de détecter une tendance de surapprentissage (basée sur le R²).

## Performances

**Les modèles linéaires**

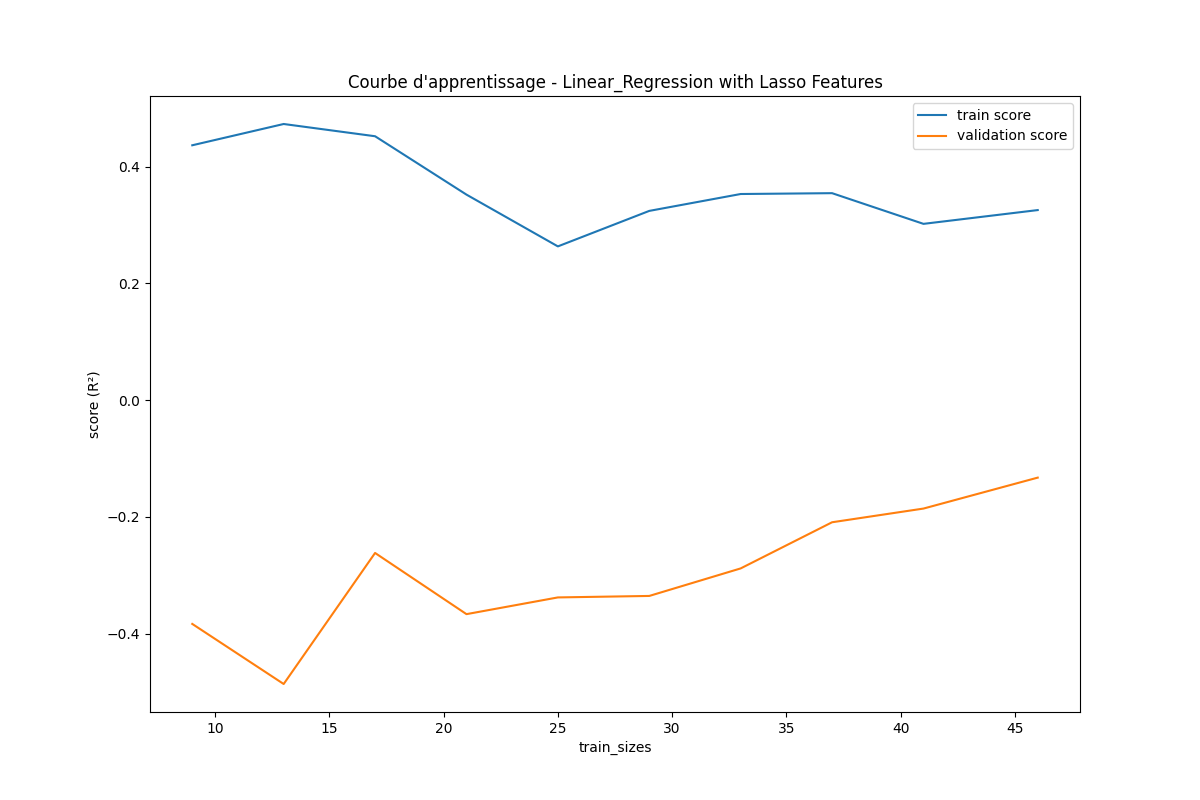
Les métriques RMSE et R² des modèles linéaires sur le jeu train et le jeu test sont indiqués dans le tableau ci-dessous :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Modèle | RMSE train | R² train | RMSE test | R² test |
| Régression linéaire – all features | 1.14 | 0.96 | 16.74 | -13.15 |
| Régression linéaire – lasso sél. | 4.58 | 0.30 | 4.61 | -0.07 |
| Régression linéaire – forward sél. | 2.80 | 0.74 | 5.09 | -0.31 |

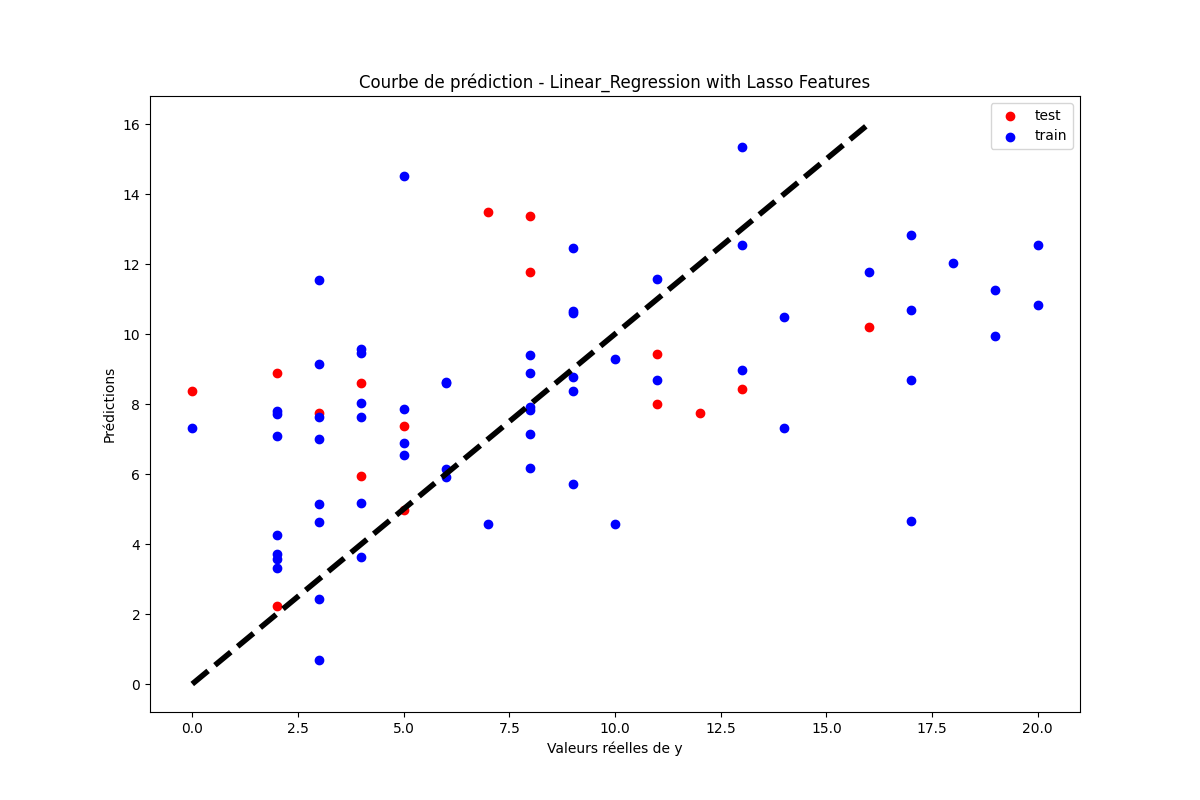
Pour le modèle linéaire de régression nous observons un très mauvais résultat incluant toutes les features. Notre modèle peut assez bien estimer les notes qu’il avait vu pendant l’apprentissage, mais généralise très difficilement à d’autres données. Il fait même pire qu’une prédiction complètement au hasard. En plus on voit que notre modèle ne va pas améliorer en ajoutant plus de données dans la phase d’apprentissage.

Ce résultat n’est pas étonnant, vue que nous avions observé en avant qu’il y avait une très forte multicollinéarité entre les features qui va impacter notre modèle.

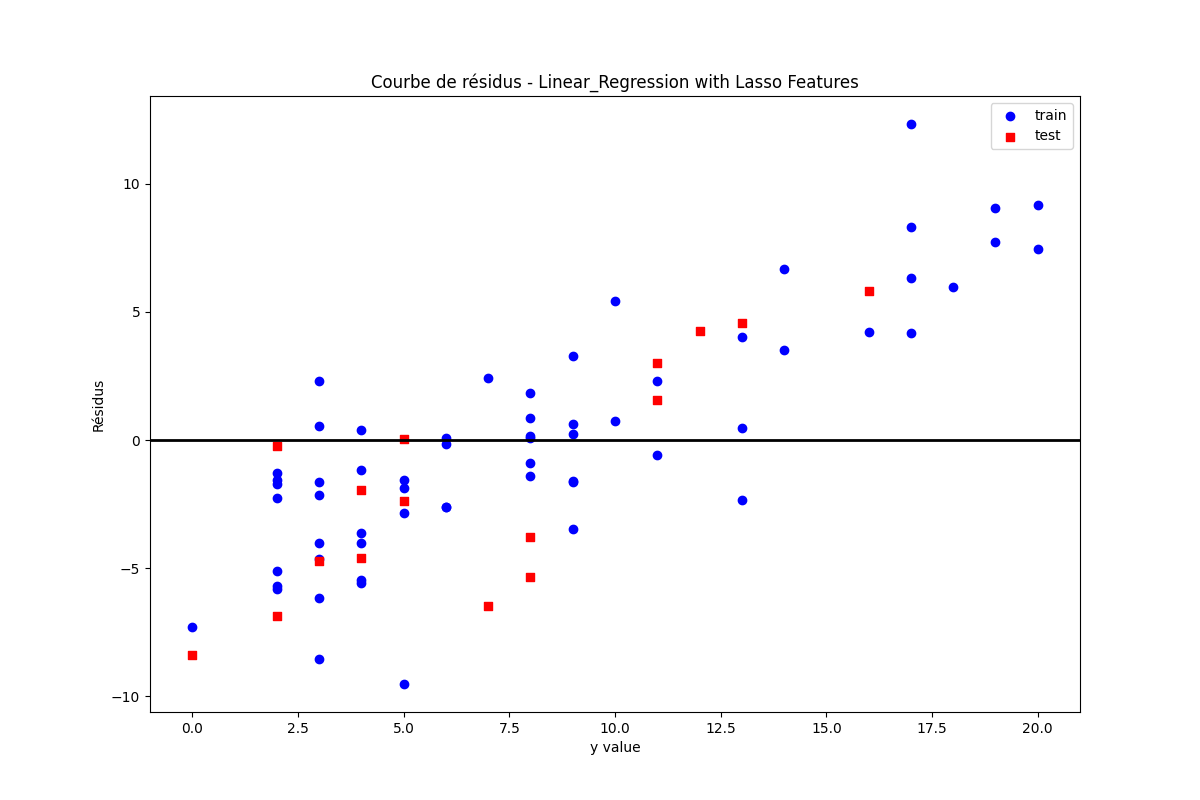
Testant un modèle linéaire avec nos deux manières de *feature selection*, on observe un comportement différent. Les performances dans le jeu d’entrainement ont fortement baissé, mais les performances dans le jeu de test augmentent. La meilleure performance pour le jeu de test est avec une sélection de features utilisant Lasso, même si le modèle ne prédit pas mieux qu’une prédiction totalement au hasard. Inclure plus de données dans le jeu d’entrainement pourra potentiellement améliorer ce modèle.



On peut observer aussi que notre modèle a tendance à prédire des valeurs autour de la moyenne. Ce comportement peut être observé pour la plupart des modèles testés.



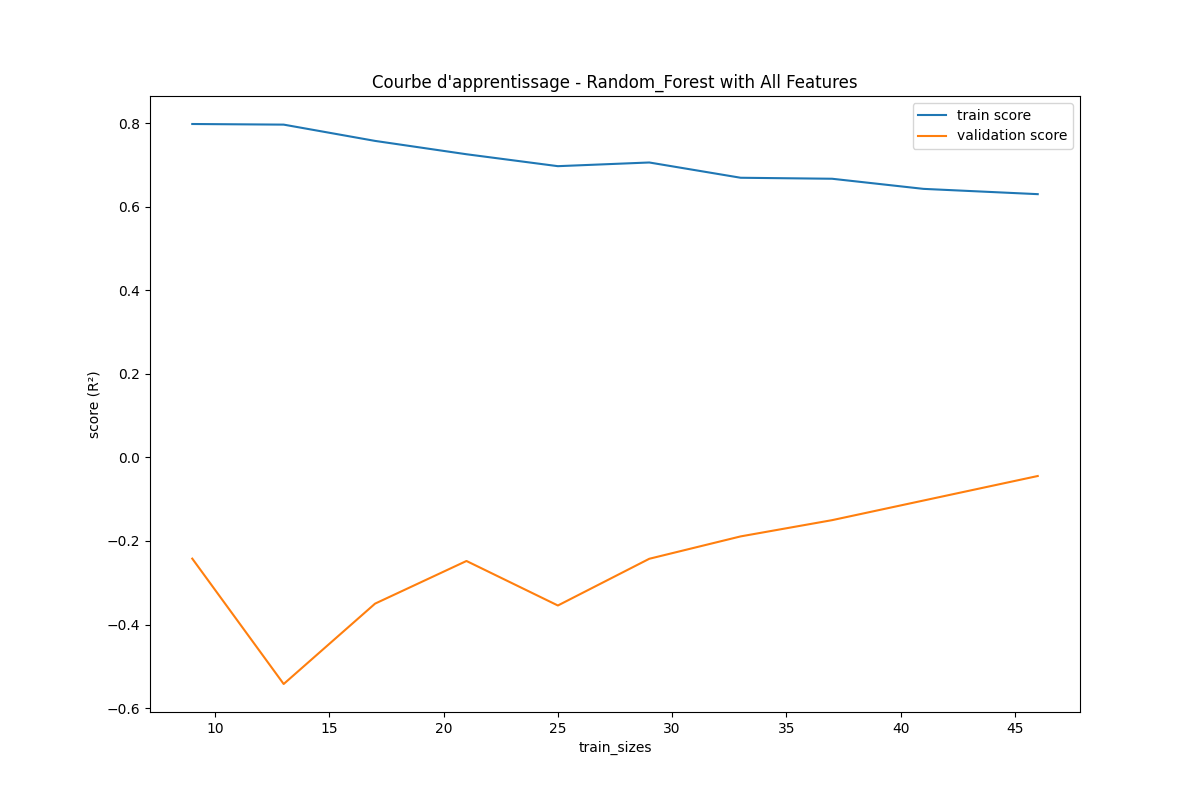
En plus on voit que, pour tous les modèles testés, les résidus augmentent quand la valeur de y augment. Les notes les mieux prédites sont ceux autour de la moyenne, ce qui n’est pas étonnant vu le constat d’avant.



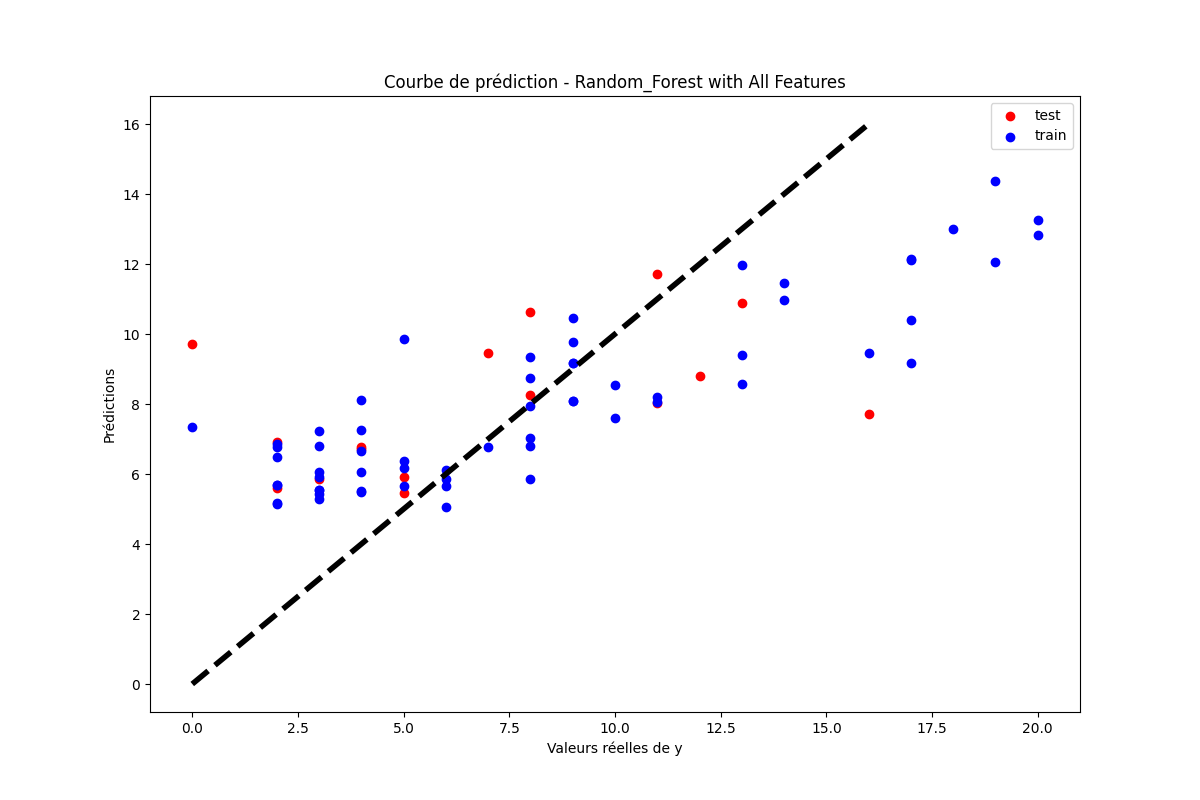
**Les modèles Random Forest**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Modèle | RMSE train | R² train | RMSE test | R² test |
| Random Forest – all features | 3.56 | 0.58 | 4.04 | 0.18 |
| Random Forest – lasso sél. | 4.32 | 0.38 | 4.21 | 0.11 |
| Random Forest – forward sél. | 3.59 | 0.57 | 4.18 | 0.12 |

Pour le modèle de régression RandomForest, on observe la meilleure performance quand on inclus toutes les features. Plus des données dans le jeu d’entrainement pourra améliorer le modèle.



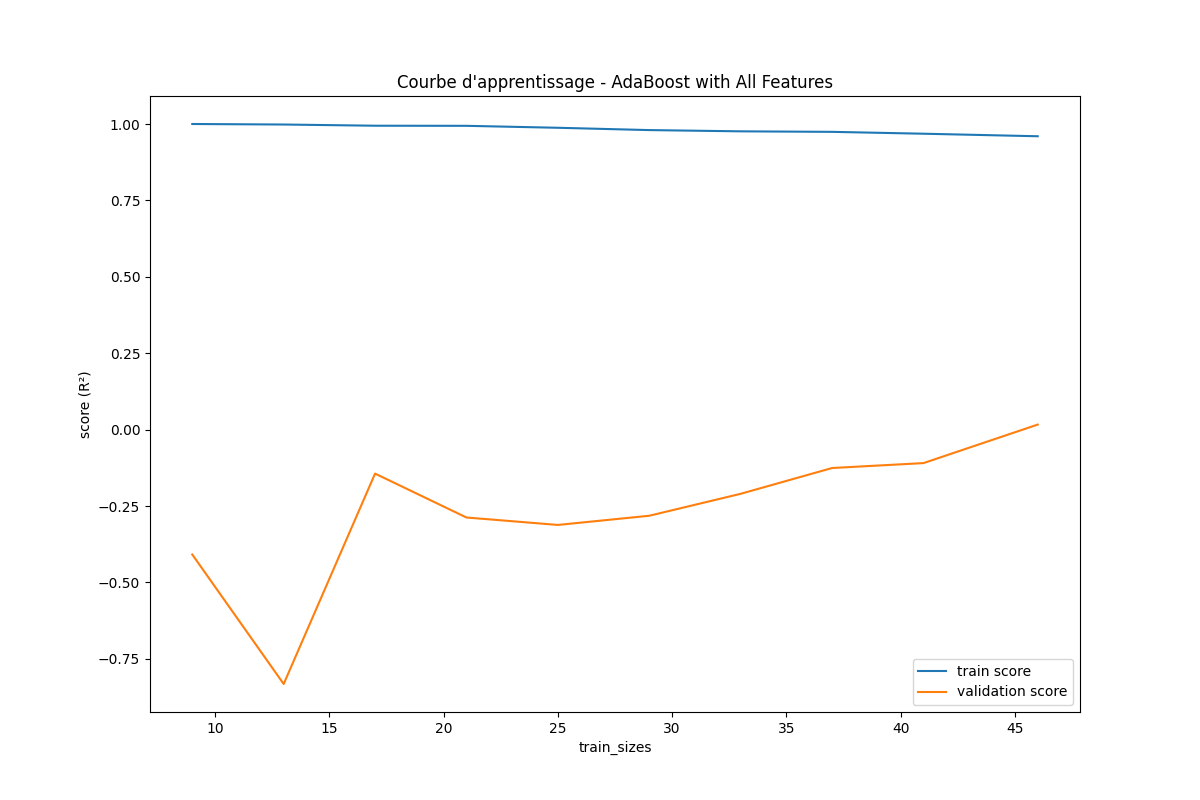
Cependant, on voit que le modèle n’est toujours pas très puissant et va surtout prédire des valeurs autour de la moyenne. Les performances sont toujours très mauvaises, surtout pour les notes bas ou hauts.

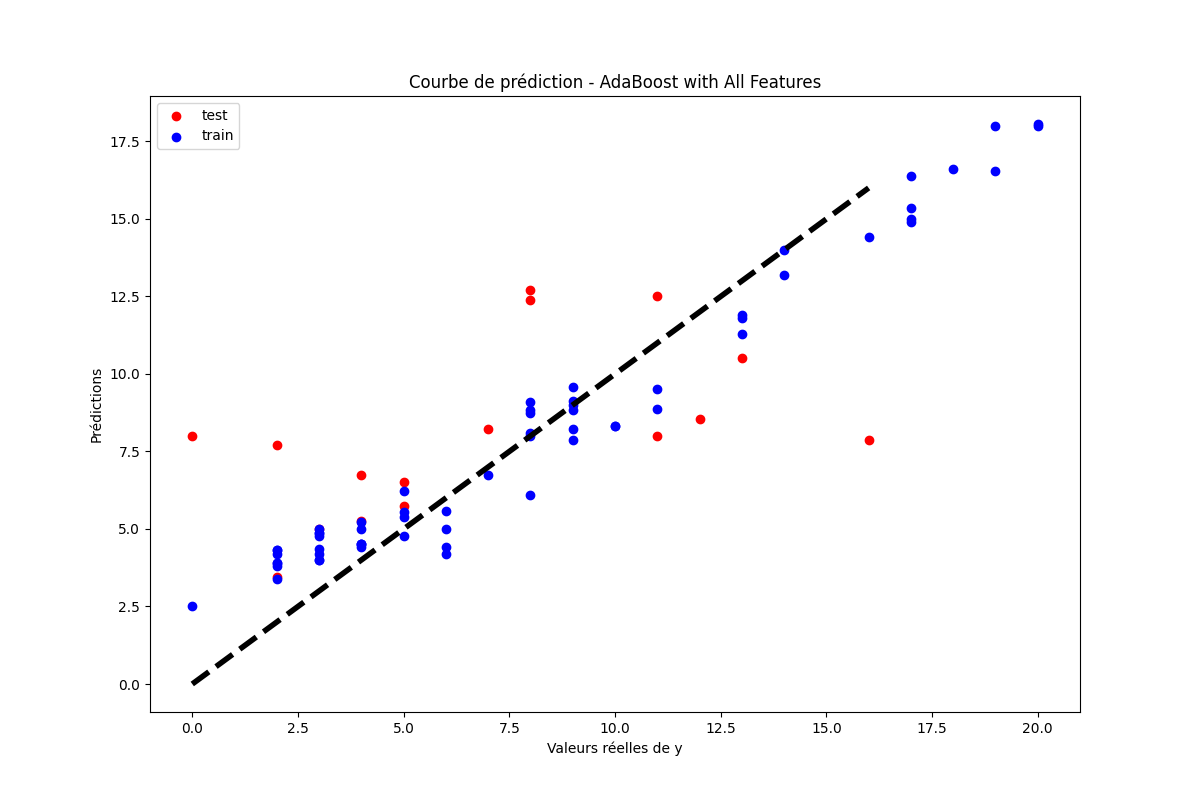


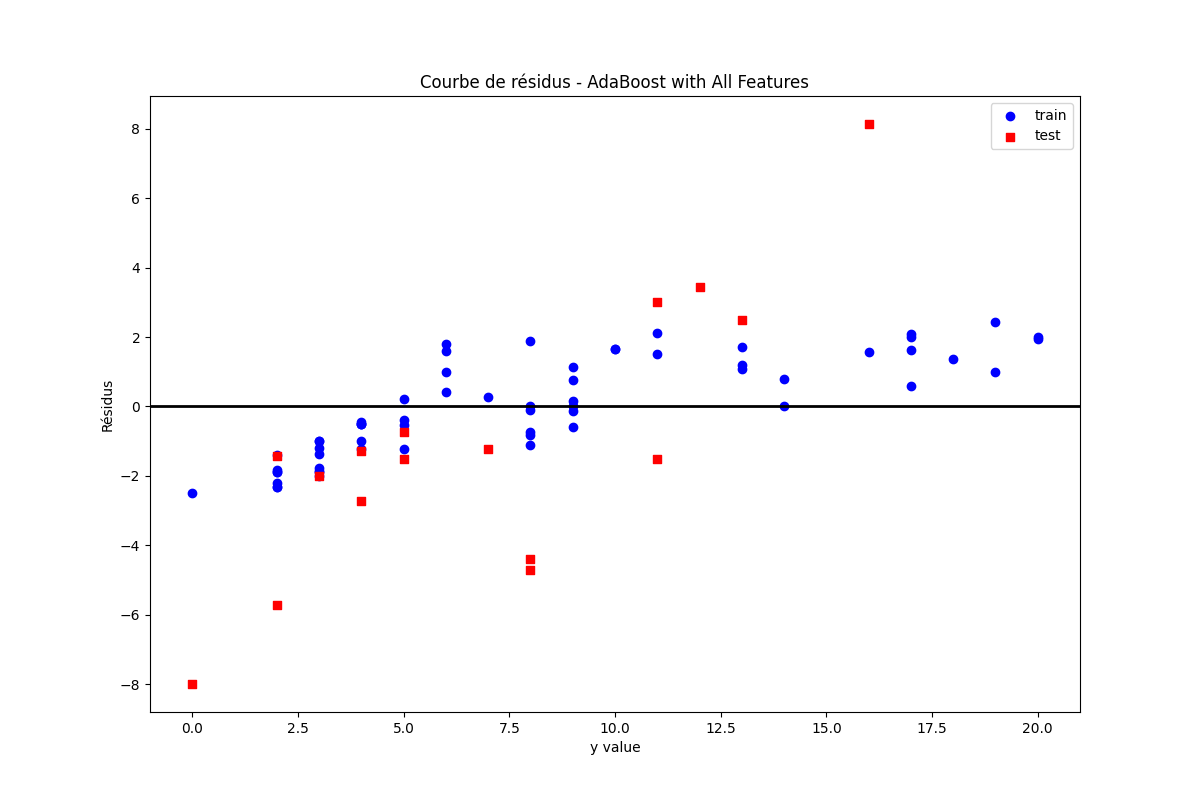
**Les modèle Adaboost**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Modèle | RMSE train | R² train | RMSE test | R² test |
| Adaboost – all features | 1.41 | 0.93 | 3.98 | 0.20 |
| Adaboost – lasso sél. | 3.34 | 0.63 | 4.26 | 0.08 |
| Adaboost – forward sél. | 1.81 | 0.89 | 4.62 | -0.08 |

Le meilleur modèle Adaboost est également celui qui inclus toutes les features. On fait les mêmes observations que le modèle antécédent, notamment que on pourra améliorer les performances en ajoutant plus de données (données de plus d’apprenants) et que le modèle prédit surtout des valeurs autour de la moyenne.







Défis rencontrés

## Généralisation des features

Lors du calcul des features, nous avons transformé les colonnes catégorielles en plusieurs colonnes, en comptant le nombre de fois que cette catégorie apparaissait. Cependant, si un nouveau fichier de logs contient des valeurs inédites dans ces colonnes, notre modèle ne reconnaît pas ces nouvelles catégories et génère une erreur. Ce problème survient car les colonnes générées pendant l'entraînement ne correspondent plus à celles du fichier de test. Il est donc nécessaire de gérer dynamiquement ces nouvelles catégories ou de s'assurer que seules les catégories connues sont utilisées lors de la transformation des données.

Nous voyons deux manières de traiter ce problème :

* Au moment de création de la DataFrame avec les features, on peut tester s’il y a une différence dans les colonnes. On peut alors supprimer les colonnes inconnues par le modèle et ajouter des colonnes qui manques (en mettant toutes les valeurs à zéro, vue qu’ils n’ont pas été observé).
* Si on a une liste de toutes les actions possibles sur la plateforme ARCHE, on peut définir une liste avec les noms de colonnes et forcer une certaine structure à notre DataFrame.

## Quantité de données

## Notre jeu de données contient environ 29 000 lignes de logs, ce qui peut sembler être un volume important. Cependant, ces données proviennent uniquement de 78 individus, ce qui représente un nombre relativement restreint pour entraîner un modèle de prédiction robuste. Pour améliorer les performances de notre modèle, il est primordial d’ajouter plus de données.

## Représentativité de données

Bien que la taille de l’échantillon joue un rôle crucial dans la capacité d’un modèle de machine learning à généraliser correctement aux nouvelles données, un grand nombre de lignes ne garantit pas nécessairement une bonne performance si la diversité des individus est limitée. Ici nous disposons de données des apprenants d’un cours sur une année. Les prédictions obtenues ne seront pas généralisables avec fiabilité aux apprenants d’autres cours et d’autres années. Pour obtenir des résultats plus généralisables, nous conseillons de prendre les données de plusieurs cours sur plusieurs années, pour limiter les effets de facteurs confoundeurs.